

Este é o cache do Google de <https://www.construct.net/en/tutorials/guide-construct-2s-advanced-158>. Ele é um instantâneo da página com uma aparência que ela tinha em 31 out. 2023 01:58:54 GMT. A [página atual](#) pode ter sido alterada nesse meio tempo. [Mais saiba mais](#).

[Versão completa](#)   [Versão somente texto](#)   [Ver código-fonte](#)

Dica: para localizar rapidamente o termo de pesquisa nesta página, pressione **Ctrl+F** ou **⌘-F**(Mac) e usar uma barra de localização.

[Tutoriais](#) > [Construção 2](#) >

Guia para os recursos avançados de eventos do Construct 2 ▼

# GUIA PARA OS RECURSOS AVANÇADOS DE EVENTOS DO CONSTRUCT 2



Ashley

Equipe Construct Fundador

publicado em 13 Abr, 2012 às 15:47



245 favoritos

## Tagged

Avançado

construção2

fluxo

## Estatísticas

Construct 2 oferece alguns recursos avançados na folha de eventos. Isso pode permitir que usuários experientes aproveitem ao máximo o sistema de eventos, permitindo uma lógica mais sofisticada do que normalmente é possível com eventos padrão. Este tutorial resume os recursos do evento que são destinados para uso avançado ou funcionam de forma diferente dos eventos normais, com algumas dicas e truques.

## Ferramentas

## Compartilhar



## Traduções

Francês

**Guia de construire de deux fonctions d'évélements avancés**

TablóideA

Português (Braziliano)

**Guide to Construct 2's recursos para eventos avançados**

Ed0Andrew

Russo

?? ?? ??

**Construir 2**

flamenon

Espanhol

**Guia avanzada de las características de los eventos para Construct 2.**

katzin

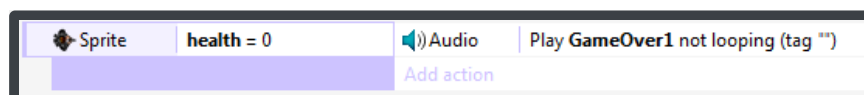
## CONDIÇÕES ESPECIAIS

O objeto System tem três condições que funcionam de forma diferente das condições normais. Para mais informações, consulte [condições do sistema](#) no manual.

### Acione uma vez enquanto é verdade

Esta condição efetivamente faz um evento continuamente verdadeiro em um gatilho. Ele testa "se não atingiu esta condição último tick". Geralmente é colocado como a última condição em um evento, porque se for a primeira condição, será alcançado imediatamente e, portanto, nunca satisfazerá seu teste.

Um dos lugares *Disparar uma vez* é útil para tocar sons. Considere o seguinte evento:

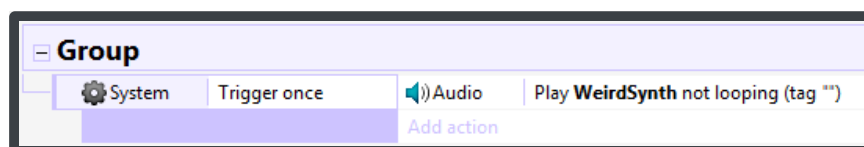


Este é um erro comum - lembre-se de que o evento é executado a cada tick, o que é cerca de 60 vezes por segundo na maioria dos computadores - e resulta no som sendo reproduzido e reproduzido continuamente, desde que a variável de saúde do jogador seja igual a 0. Muitas vezes o resultado parece horrível. Em vez disso, queremos reproduzir um único efeito sonoro na primeira vez em que o contador de saúde for igual a 0. Adicionando *Disparar uma vez* consegue isso:

Este tutorial está licenciado sob [CC BY-NC-SA 4.0](#). Por favor, consulte o texto da licença se você deseja reutilizar, compartilhar ou remixar o conteúdo contido neste tutorial.

atinge 0. Não vai jogar novamente até saúde ser alterada para outra coisa, em seguida, volta para 0, quando ele vai tocar um único som novamente.

Um truque útil é colocar *Disparar uma vez por si só* em um grupo de eventos:



Se o grupo estiver sendo ativado e desativado durante o jogo, ele executará o evento uma vez quando o grupo estiver ativado. Ele atua como uma espécie de gatilho "On group enabled".

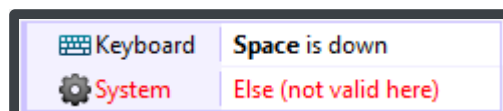
## Escolha tudo

Conforme descrito em *Como os Eventos Funcionam*, eventos normalmente funcionam por *filtragem* instâncias que não atendem às condições. Um subconjunto de instâncias é deixado que atendem a todas as condições e, em seguida, as ações são executadas nessas instâncias.

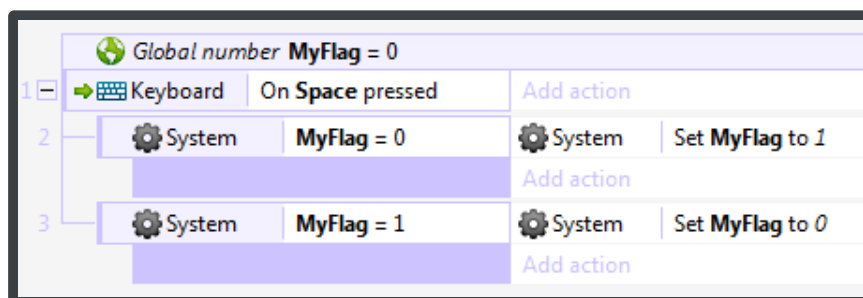
*Escolha tudo* é a única condição que funciona ao contrário: restaura todos os objetos, de modo que as condições subsequentes selecionam todas as instâncias novamente. É difícil chegar a um exemplo simples e intuitivo disso, mas pode ser útil para usuários avançados que precisam lidar com subeventos profundamente aninhados, onde é conveniente redefinir os objetos escolhidos.

## Outro

aparecerá vermelho indicando que você deve movê-lo, como mostrado abaixo. Você não pode visualizar ou exportar seu projeto se *Outro* as condições estão no lugar errado, já que a lógica não faz sentido.

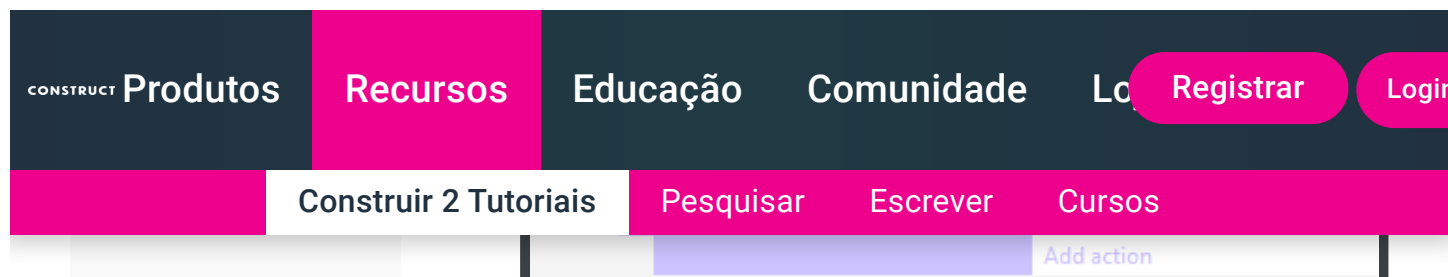


Um lugar comum *Outro* é útil quando se alterna um sinalizador ou variável. Muitas vezes, novos usuários cometem o seguinte erro ao alternar uma variável, neste caso, ao pressionar Espaço:



Observe como se *MyFlag* é 0 é definido como 1, mas então o próximo evento imediatamente o define de volta para 0 novamente, porque *MyFlag* é igual a 1! (Este é o lugar onde a ordem dos eventos é importante: lembre-se de eventos executados de cima para baixo.) Este evento não tem o efeito pretendido e *MyFlag* permanece 0, não importa quantas vezes o espaço é pressionado.

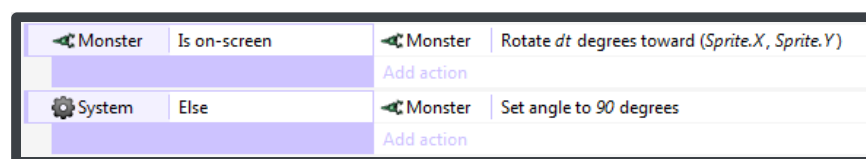
Uma solução é definir *MyFlag* para 1 - **MyFlag**. No entanto, isso só funciona para números e nem sempre é muito legível. *Outro* pode resolver o problema:



Agora pressionando Espaço corretamente alterna *MyFlag* entre 0 e 1, porque o *Outro* só é executado se o evento anterior não.

Pressionando **X** é um **atalho de teclado** para adicionar um *Outro* evento após o evento selecionado no momento.

Nota *Outro* faz **não** escolha quaisquer objetos. Significa literalmente apenas "o último evento não foi executado". Considere o seguinte exemplo:



A intenção pode ser fazer com que os monstros na tela girem em direção ao jogador, e fazer o resto apontar para baixo a 90 graus. Há dois problemas com isso. Em primeiro lugar, o *Outro* o evento não será executado se *qualquer* monstros estão na tela, desde então o primeiro evento foi executado para que o *Outro* não corre. Em segundo lugar, mesmo que o *Outro* o evento é executado, ele não escolhe especificamente monstros que estão fora da tela: afeta *tudo* monstros, porque não escolhe instâncias. Neste caso, é melhor simplesmente substituir o *Outro* com um invertido *Está na tela* condição, como mostrado abaixo. Isso terá o efeito pretendido nas instâncias dentro e fora da tela.

*Outro* também pode ser acorrentado em "*Else-if*" blocos adicionando condições extras ao *Outro* evento e, em seguida, adicionar outro *Outro* evento depois disso. Isso é mostrado abaixo.

2	System	ItemCount = 0	Text	Set text to "You have no items!"
			Add action	
3	System	Else	Text	Set text to "You have one item!"
	System	ItemCount = 1	Add action	
4	System	Else	Text	Set text to "You have lots of items!"
			Add action	

Isso pode ser lido:

Se *ItemCount* is 0: definir texto para "Você não tem itens!"

Outra se *ItemCount* is 1: definir texto para "Você tem um item!"

Else: defina o texto para "Você tem muitos itens!"

Isso imita outras cadeias em linguagens de programação.

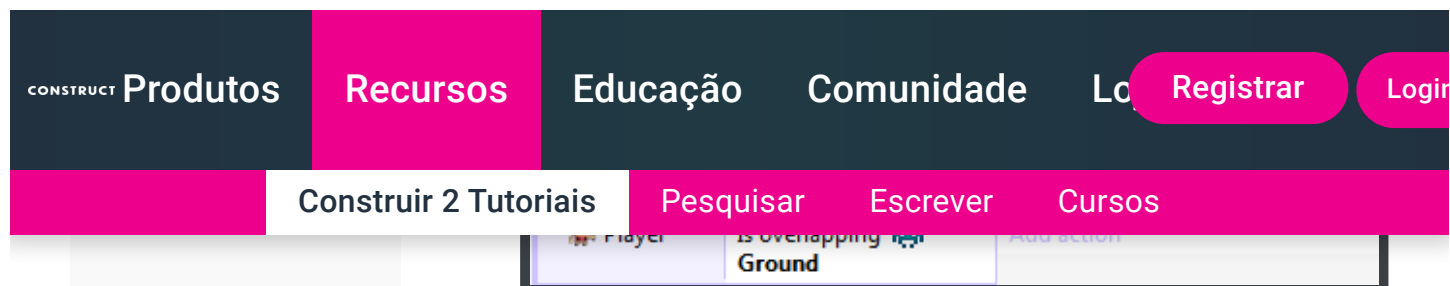
## ENQUANTO LOOPS

Outra condição interessante do sistema é o *Enquanto* loop. Os outros loops (repetir, para, para-cada) são relativamente simples, mas *Enquanto* funciona de forma ligeiramente diferente e por isso merece sua própria menção.

*Enquanto* executa o evento infinitamente até que:

uma condição que se segue torna-se falsa, ou

- o *Parar loop* a ação do sistema é usada.

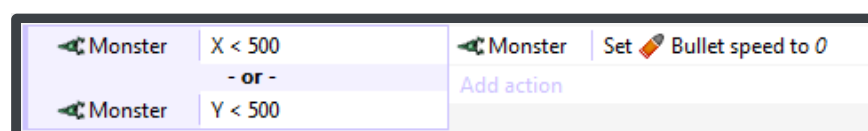


Isso continuará movendo o jogador 1 pixel até que ele não esteja mais sobrepondo 'Ground'. Isso acontece instantaneamente, então repetirá quantas vezes for necessário até que 'Is overlapping Ground' se torne falso.

Tenha cuidado para não criar loops infinitos acidentalmente. Se a condição nunca se torna falsa durante o loop, ou a condição 'Enquanto' é usada por conta própria sem uma ação 'Stop loop', ela será suspensa enquanto loops para sempre.

## 'BLOCOS 'E' VS. BLOCOS 'OR'

Eventos normais usam a lógica 'And: *tudo* as condições devem ser atendidas para que as ações sejam executadas. Em outras palavras, "condição 1 E condição 2 E condição 3.." deve ser verdade. Em contraste, os blocos 'Or' funcionam para *qualquer* de suas condições que são verdadeiras. Em outras palavras, eles são executados se "condição 1 OU condição 2 OU condição 3.." são verdadeiras. Considere o seguinte exemplo:

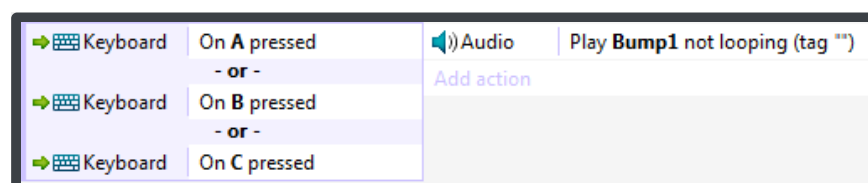


A intenção aqui é parar qualquer monstro que reste de  $X=500$  ou acima de  $Y=500$ . Nesse caso, se alguma instância atender a qualquer condição, o evento será executado. As instâncias escolhidas

condições subsequentes *remover* instâncias que não atendem ao evento - progressivamente *reduzindo* o número de instâncias escolhidas. Em contraste, blocos 'OR *adicionar* instâncias que atendem ao evento - progressivamente *aumentando* o número de instâncias escolhidas.

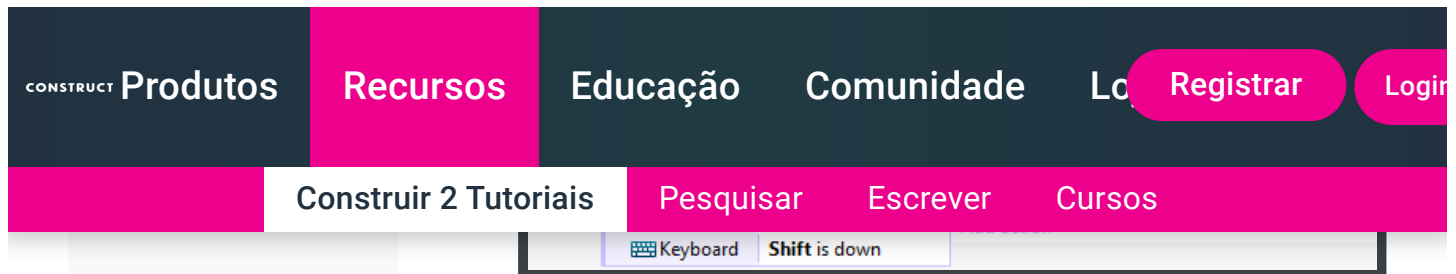
Por padrão, os blocos de eventos são do tipo 'E' normal. Eles podem ser alternados de e para blocos 'Ou' clicando com o botão direito do mouse na margem do evento e selecionando *Faça o bloco 'Or'*, ou pressionando o **Y** *atalho de teclado*.

Normalmente, apenas um gatilho pode ser colocado em um evento. No entanto, isso não se aplica aos blocos 'Ou'. Vários gatilhos podem ser adicionados a um evento 'Ou' e as ações serão executadas quando qualquer um dos gatilhos no evento for disparado. Por exemplo, abaixo mostra um evento que reproduz um som se as teclas A, B ou C no teclado forem pressionadas:



Observe que, como os blocos normais ('And') não podem conter vários gatilhos, esse evento não pode ser retornado para um bloco 'And' até que todos, exceto um dos gatilhos, sejam excluídos.

Usando subeventos, você pode combinar blocos 'Or' com blocos 'And' para criar uma lógica mais avançada. Por exemplo:



Isso tocará um som ao pressionar A, B ou C, desde que o Control ou o Shift também sejam mantidos pressionados. Alternativamente: "Se (A pressionada OU B pressionada OU C pressionada) E (Controle está para baixo OU Shift está para baixo): tocar som".

Se o segundo evento não fosse um bloco 'Or', ele leria "Control is down AND shift is down". Portanto, um som seria reproduzido se você mantiver pressionado Control + Shift e pressionar A, B ou C. Alternativamente: "Se (A pressionada OU B pressionada OU C pressionada) E (O controle está inativo E o Shift está inativo): reproduza o som". Portanto, usar subeventos pode ajudá-lo a criar uma lógica condicional mais avançada.

## OUTROS RECURSOS AVANÇADOS ÚTEIS

### IDs únicos (UIDs) e IDs de instância (IIDs)

UIDs e IIDs são frequentemente úteis para usuários avançados para seleção avançada de instâncias.

Um UID pode ser usado como uma "referência" a um objeto. Um UID pode ser armazenado em uma variável e o objeto mais tarde escolhido com o *Escolha por ID único* condição.

IIDs podem ser usados para recuperar expressões de instâncias específicas. Por exemplo,

[expressões](#). IIDs também podem ser usados para escolher instâncias usando o *Escolha a instância Nth* condição do sistema, mas cuidado que um IID não é uma referência permanente a um objeto: para que um UID deve ser usado em seu lugar.

Para mais informações, consulte [UIDs e IIDs no manual](#).

## O MAIS AVANÇADO DE TODOS...

Se você tem experiência de programação anterior ou apenas quer levá-lo para o próximo nível, você pode usar o [SDK Javascript](#) para escrever seus próprios plugins e comportamentos para o Construct 2. O [visão geral](#) page tem uma lista de links para ajudá-lo a aprender Javascript.

### 1 COMENTÁRIOS

Ordem por

Melhor



Quer deixar um comentário? [Login](#) ou [Registrar uma conta](#)!



**abdo html5** 1 Pontos 4 Anos atrás

bom tut

[Permalink Responder](#)

## PRODUTOS

Software de Criação de Jogos  
Software Animação

CONSTRUCT

Produtos

Recursos

Educação

Comunidade

Lo

Registrar

Login

Construir 2 Tutoriais

Pesquisar

Escrever

Cursos



**A SUA LOCALIZAÇÃO**

Estados Unidos

**NOSSA EMPRESA**

Sobre Nós

Nossa Equipe

Kit de Imprensa

Na Imprensa

Entre em Contato

Faça Seu Próprio Jogo

Preços

Documentação

FAQ

Estudos de Caso